ซอฟต์แวร์
เทอร์โบปาสคาลกับวินโดวส์
: ไดอะล็อกบ็อกซ์ (2)

เทอร์โบ-ปาสคาลกับวินโดวส์
: ไดอะล็อกบ็อกซ์ (2)

การสร้างรีซอร์สและการเขียนโปรแกรมเพื่อเปรียบเทียบกับการเขียนโปรแกรมที่ใช้คอนโทรลเป็นออปเจ็กต์

ในตอนที่แล้วผมได้แสดงการสร้างรูปแบบของไดอะล็อกบ็อกซ์ คือ รีซอร์สไว้แล้ว เพื่อจะนำมาเขียนโปรแกรมในตอนนี้ แต่คงมีเรื่องที่ต้องนำมากล่าวถึงก่อนครับ

ทำความเข้าใจเพิ่ม

รูปแบบไดอะล็อกบ็อกซ์ของนามบัตร เหมาะที่จะนำมาประกอบคำอธิบายในการสร้างรีซอร์ส และการเขียนโปรแกรมเพื่อให้เปรียบเทียบกับการเขียนโปรแกรมที่ใช้คอนโทรลเป็นออปเจ็กต์ แต่ก็ไม่เหมาะที่จะใช้ประกอบคำอธิบายตามที่ผมจะกล่าวต่อไปนี้ ขอให้สร้างรูปแบบของไดอะล็อกบ็อกซ์ใหม่ตามรูปที่ 1 (เฉพาะส่วนที่เป็นไดอะล็อกบ็อกซ์) โดยให้ค่าไอดีของปุ่ม OK เป็น 1 ปุ่ม Cancel เป็น 2 ปุ่ม Stop เป็น 101 กรอบอีดิตเป็น 201 และกรอบหัวข้อเป็น 202 เก็บไว้ในไฟล์ TSTDLG1.RES โดยให้ชื่อรีซอร์สเป็น 200

โมเดลไดอะล็อกบ็อกซ์

พร้อมทั้งสร้างรูปแบบของเมนูตามรูปที่ 1 เช่นกัน โดยให้ค่ารายการ Demo เป็น 101 และ Start เป็น 102 เก็บไว้ในไฟล์เดียวกัน (ใช้ Save To) ใช้ชื่อรีซอร์สเป็น 100

จุดประสงค์ของผมก็เพื่อที่จะแสดงทางเดินของโปรแกรมเมื่อผู้ใช้เลือกปุ่มต่าง ๆ เช่น OK หรือ Cancel ก็ต้องมีโปรแกรมสนับสนุนครับ ขอให้ลอกโปรแกรมตามรูปที่ 5 ในไมโครฯ 87 ไปไว้ในไฟล์ TSTDLG2.PAS แล้วปรับปรุงโปรแกรมดังนี้ครับ

1. แก้ชื่อไฟล์ .RES และการกำหนดค่าคงตัวใหม่ดังรูปที่ 2

2. แก้ชื่อรีซอร์สจาก 'nmecrdmenu' เป็น menuname และจาก 'NMECRDDLG' เป็น dlgname คือชื่อใหม่นี้เป็นค่าคงตัวตามรูปที่ 10 จึงไม่ต้องอยู่ในเครื่องหมายคำพูด

3. ป้อนโปรแกรมการกำหนดสกุล TstDlgC ตามรูปที่ 3 และตัวโปรแกรมในรูปที่ 4

4. แก้ชื่อสกุลในคำสั่ง New ของ WndProC.cmDemo จาก PDialog เป็น TstDlgC

ก็จะได้โปรแกรมสำหรับทำความ

เข้าใจทางเดินของโปรแกรมของโมเดลไดอะล็อกบ็อกซ์ ซึ่งเมื่ออ่านจากคู่มือแล้วอาจสับสน ขอให้ทดลองรันและเลือกปุ่มต่าง ๆ ดูจะได้ผลดังนี้ครับ

– เลือกปุ่ม OK จะไปทำงานที่ OK, Canclose และ Done

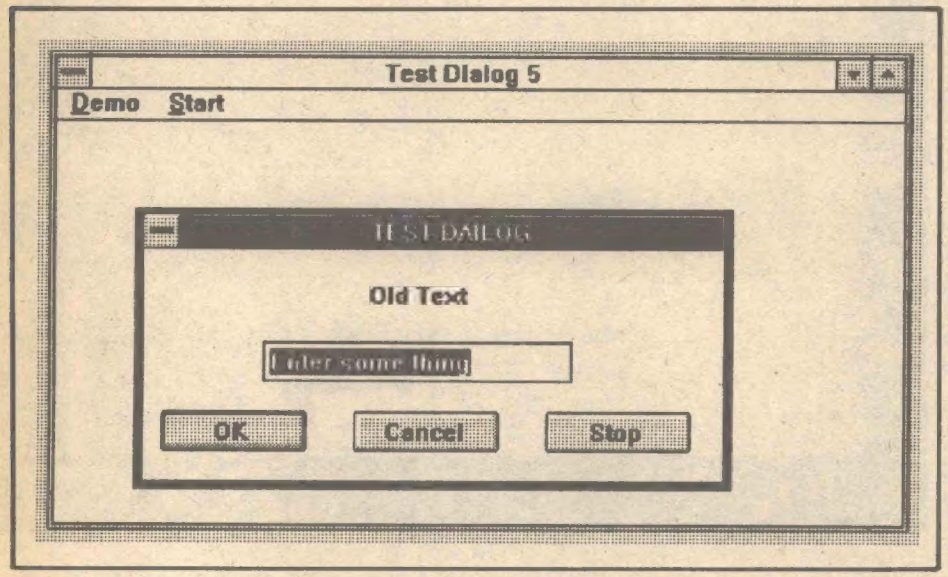
– เลือกปุ่ม Cancel หรือเลือกรายการ Control/Close ของไดอะล็อกบ็อกซ์ จะทำงานที่ Cancel และ Done

– เลือกปุ่ม Stop จะทำงานที่ idStop และ Done

เป็นไปตามข้อกำหนดของวินโดวส์ที่ให้ผู้ใช้ เลือกปุ่ม OK เมื่อต้องการให้นำสิ่งที่ตนป้อนหรือเลือก หรือเซตค่าไว้ในไดอะล็อกบ็อกซ์ไปดำเนินการ และยกเลิกเมื่อเลือกปุ่ม Cancel หรือรายการ Control/Close

ซึ่งเราจะต้องเขียนโปรแกรมตามนี้ด้วยครับ คือ เขียนโปรแกรมที่จะให้มีการดำเนินการไว้ใน OK และยกเลิกหรือไม่ต้องเขียนโปรแกรมใน Canclose

สำหรับปุ่ม เช่น Stop ปกติไม่มีเขียนไว้เพื่อแสดงว่าถ้าต้องการยุติการทำงานของไดอะล็อกบ็อกซ์ ด้วยตัวเราจะดำเนินการได้อย่างไรคือที่ใช้คำสั่ง EndDlg (id_stop) ค่าที่ให้คือ id_stop จะให้เป็นค่าใดก็ได้ และจะเป็นค่าส่งกลับของฟังก์ชัน ExecDialog



รูปที่ 1 หน้าต่างหลักของไดอะล็อกบ็อกซ์ สำหรับทำความเข้าใจกับไดอะล็อกบ็อกซ์

```
{$R tstdlg1.RES}

CONST
  dlgname    = 200;    menuname   = 100;
  cm_demo    = 101;    cm_start   = 102;
  id_stop    = 101;    id_anedit  = 201;   id_atext = 202;
```

รูปที่ 2 รีซอร์สและค่าคงตัวที่กำหนดใหม่

โหมดเลสส์ไดอะล็อกบ็อกซ์

การสร้างโหมดเลสส์ไดอะล็อกบ็อกซ์ คงดำเนินการเช่นกัน แต่เมื่อสร้างรูปแบบด้วยไดอะล็อกบ็อกซ์เอดิเตอร์ ให้เซตปุ่ม Visible ไว้ดังรูปที่ 6 ปุ่มนี้ถึงเซตไว้ก็ใช้กับโมเดลไดอะล็อกบ็อกซ์ได้ไม่มีปัญหาใด

อีกสองปุ่มที่อาจต้องการใช้คือปุ่ม Minimize Box กับปุ่ม Maximize Box หากเซตสองปุ่มนี้จะได้ไดอะล็อกบ็อกซ์ที่มีปุ่มทั้งสองนี้ และมีรายการอยู่ในคอนโทรลเมนู แต่โดยทั่วไปไม่ใช้กัน สำหรับปุ่มรายละเอียดอื่น ๆ ขอให้ดูในหัวข้อ ds_ และ ws_ สำหรับของคอนโทรลให้ดูในหัวข้อที่ขึ้นต้นด้วยชื่อคอนโทรลตามด้วย s เช่น bs__ เป็น button style

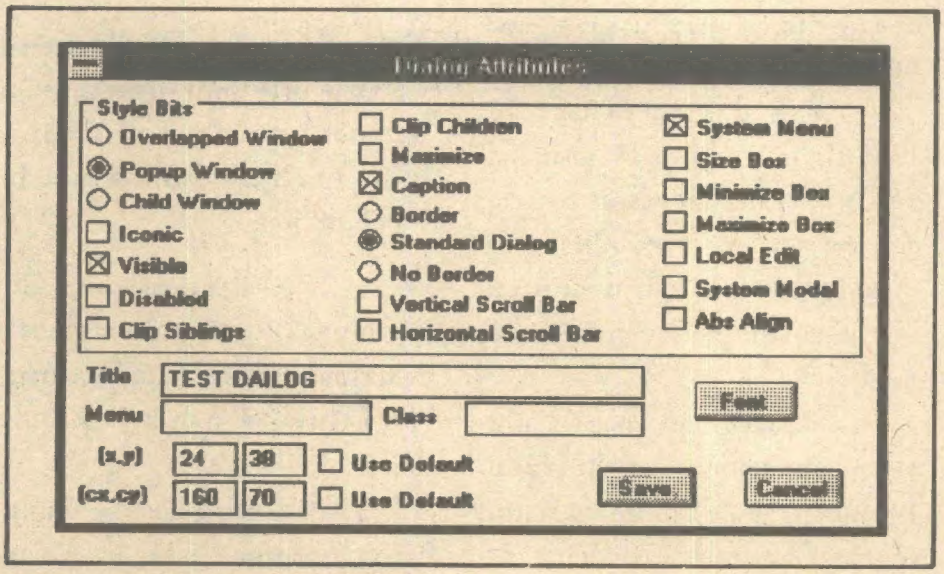
เมื่อรันเพราะโหมดเลสส์ไดอะล็อกบ็อกซ์ไม่ได้ครอบครองอินพุตตลอดเวลา ผู้ใช้จึงสามารถเลือกรายการ Demo ได้อีกเปิดไดอะล็อกบ็อกซ์ได้อีกจนกว่าหน่วยความจำจะไม่พอ ในโปรแกรมจึงควรจำกัดไว้เช่นถึงจำนวนที่กำหนดให้ดิสเอเบิลรายการ Demo

รูปที่ 3 สกุล TstDlgC

```
 TstDlgCP = ^TstDlgC;
 TstDlgC  = object(TDialog)
constructor  Init(apar : PWindowsObject; atitle : PChar);
destructor   Done; virtual;
function     CanClose : Boolean; virtual;
procedure    Ok(var msg : TMessage); virtual id_first + id_ok;
procedure    Cancel(var msg : TMessage); virtual id_first + id_cancel;
procedure    idStop(var msg : TMessage); virtual id_first + id_stop;
             end;
```

```
CONSTRUCTOR  TstDlgC.Init(apar : PWindowsObject;
                          atitle : PChar);
BEGIN
  TDialog.Init(apar,atitle);
END;


DESTRUCTOR  TstDlgC.Done;
BEGIN
  MessageBox(HWindow,'DONE','',mb_Ok);
  TDialog.Done
END;


FUNCTION  TstDlgC.CanClose : Boolean;
BEGIN
 CanClose := True;
 MessageBox(HWindow,'CANCLOSE',nil,mb_Ok);
END;


PROCEDURE  TstDlgC.Ok(var msg : TMessage);
BEGIN
  MessageBox(HWindow,'OK','User Select',mb_Ok);
  TDialog.Ok(msg);
END;


PROCEDURE  TstDlgC.Cancel(var msg : TMessage);
BEGIN
  MessageBox(HWindow,'CANCEL','User Select',mb_Ok);
  TDialog.Cancel(msg);
END;


PROCEDURE  TstDlgC.idStop(var msg : TMessage);
BEGIN
  MessageBox(HWindow,'STOP','User Select',mb_Ok);
  EndDlg(id_stop);
END;
```

รูปที่ 4 โปรแกรมของสกุล TstDlgC



รูปที่ 5 การให้ค่ารายละเอียดของไดอะล็อกบ็อกซ์ ตามรูปที่ 1

แต่ก็มีปัญหาเหมือนกันครับ เพราะเมื่อผู้ใช้ปิดหน้าต่างไดอะล็อกบ็อกซ์ เราไม่ทราบ ต้องให้ไดอะล็อกบ็อกซ์คือควรจะเป็นที่ CanClose ส่งเมสเสจมาให้หน้าต่างแม่ด้วย โดย SendMessage ตามที่ได้แสดงวิธีใช้ไว้ในไมโครฯ 84 หัวข้อให้หน้าต่างคุยกัน

สำหรับทางเดินของโปรแกรมจะเป็นดังนี้

– เลือกปุ่ม OK จะทำงานที่ OK, CanClose และ Done

– เลือกปุ่ม Cancel หรือรายการ Control/Close ของไดอะล็อกบ็อกซ์ จะทำงานที่ Cancel, CanClose และ Done

– เลือกปุ่ม Stop จะทำงานที่ CanClose และ Done

– เลือกรายการ Contral/Close ของหน้าต่างหลัก จะทำงานที่ CanClose เท่านั้น

เป็นไปตามข้อกำหนดของวินโดวส์ที่จะเป็นปัญหาคือ จาก Close ของหน้าต่างหลักไม่ทำงานที่ Done ซึ่งเป็นเรื่องของทางด้านเทอร์โบปาสคาล เป็นที่สำหรับเขียนโปรแกรมทางด้านโอโอพี เช่น ให้ยกเลิกหน่วยความจำที่จองไว้

หากเห็นว่าเป็นหน้าต่างหลักเป็นการยุติการทำงานของแอพพลิเคชันนั้น วินโดวส์ควรจะเรียกหน่วยความจำที่จัดสรรให้คืนก็ไม่ต้องดำเนินการใด แต่ถ้าเห็นว่าอาจมีปัญหาก็ควรจะดิสเอเบิลรายการนี้ไว้ตามโปรแกรมในรูปที่ 6

ไดอะล็อกบ็อกซ์อย่างหน้าต่าง

เฉพาะในเทอร์โบปาสคาลได้สร้างสกุล TDlgWindow ไว้ สืบสกุลจาก TDialog อีกทอดหนึ่งเป็นไดอะล็อก-บ็อกซ์กึ่งหน้าต่าง คือ มีคุณสมบัติของ

ไดอะล็อกบ็อกซ์ทุกประการ เช่น ต้องมีรีซอร์ส แต่ให้ใช้งานอย่างหน้าต่างได้ด้วย (แต่ไม่สมบูรณ์) เช่น ให้มีเคอร์เซอร์คือพอยน์เตอร์ของเม้าส์เป็นของตนเองได้ เช่น อาจทำเป็นรูปตัว I

โดยที่ในคู่มือของเทอร์โบปาสคาลแนะนำให้ใช้เป็นหน้าต่างหลักของแอพพลิเคชันที่ไม่มีหน้าต่างรอง และมีคอนโทรลเป็นจำนวนมากในหน้าต่างหลัก เช่นเครื่องคิดเลข

แต่สามารถนำมาใช้เป็นไดอะล็อกบ็อกซ์ได้ โดยดำเนินการสร้างเช่นเดียวกับโหมดเลสส์ไดอะล็อกบ็อกซ์ และสำหรับโปรแกรมมีที่แตกต่างกันเพียงแห่งเดียวคือ ให้เปลี่ยนการสืบสกุลของ TstDlgC จาก TDialog เป็น TDlo-Window เท่านั้นครับ

เมื่อรันดู ผมไม่เห็นมีอะไรแตกต่างไปจากโหมดเลสส์ไดอะล็อกบ็อกซ์ รวมทั้งทางเดินของโปรแกรมด้วย

```
PROCEDURE  WndProC.cmDemo(var msg : TMessage);
VAR    dlgp : PDialog;    Hctrl : HMenu;
       P : PWindowsObject;
BEGIN
{  dlgp := New(TstDlgCP, Init(@Self, PChar(dlgname)));  }
  dlgp := New(PDialog, Init(@Self, PChar(dlgname)));
  dlgp^.EnableAutoCreate;
  P := New(NumEditCP, InitResource(dlgp,id_anedit,txtlen+1));
  Application^.MakeWindow(dlgp);
  Hctrl := GetSystemMenu(dlgp^.HWindow,false);
  AppendMenu(Hctrl,mf_string,cm_Exit,'E&xit');
  EnableMenuItem(Hctrl,1,mf_ByPosition or mf_Grayed);
END;
```

รูปที่ 6 โปรแกรมสร้างไดอะล็อกบ็อกซ์ และได้แสดงวิธีเพิ่มรายการ Exit และคิสเอเบิล รายการ Close ในคอนโทรลเมนู

```
VAR  Hdlg, Hctrl : Hwnd;
     msg, wParam : Word;  lParam : LongInt;

BEGIN
  ........

  Hdlg := dlgp^.HWindow;
  Hctrl := GetDlgItem(Hdlg, id_name);
  SendMessage(Hctrl, msg, wParam, lParam);


  SendDlgItamMessage(Hdlg, id_name, msg, wParam, lPaaram);


  TDialog.SendDlgItemMsg(id_name, msg, wParam, lParam);
```

รูปที่ 7 แสดงวิธีส่งเมสเสจไปให้คอนโทรลในไดอะล็อกบ็อกซ์

สื่อสารกับคอนโทรล

ตามที่กล่าวมาก็เพื่อเป็นการแนะนำให้รู้จักไดอะล็อกบ็อกซ์ทั้งสามแบบในเรื่องของความแตกต่างกัน ส่วนการเขียนโปรแกรมต่อไปจะเหมือนกันครับ

การสื่อสารกับคอนโทรลเป็นการสื่อสารสองทาง คือ จากคอนโทรล (ความจริงคือวินโดวส์) มายังไดอะล็อกบ็อกซ์ เมื่อผู้ใช้ดำเนินการกับคอนโทรล ผมขอย้ำว่าจากคอนโทรลมายังไดอะล็อกบ็อกซ์เท่านั้น ซึ่งเราจะดำเนินการได้โดยดักจับเมสเสจ

ส่วนอีกทางหนึ่งคือจากหน้าต่างหรือไดอะล็อกบ็อกซ์ไปยังคอนโทรล ซึ่งจะดำเนินการได้โดยส่งเมสเสจไปให้ และทำได้หลายวิธีดังแสดงในรูปที่ 7 โดยที่

- dlgp เป็นพอยน์เตอร์ไปยังไดอะล็อกบ็อกซ์ ได้มาจากการสร้างไดอะล็อกบ็อกซ์ (เช่นในรูปที่ 6 ของตอนที่แล้ว)

- msg เป็นค่าเมสเสจที่จะส่งไปให้คอนโทรล ทำเป็นค่าคงตัวของแต่ละคอนโทรลไว้ขึ้นต้นด้วยอักษร bm, cb, dm, em และ lb สำหรับปุ่มวิทยุกับปุ่มเช็ค กรอบคอมโบ ปุ่มควบคุม กรอบอีดิต และกรอบรายการตามลำดับ เช่น cb_InsertString

- wParam และ lParam เป็นขยายความของเมสเสจ เมื่อไม่ใช้นิยมให้ค่าเป็น 0 เช่น จากเมสเสจข้างบน wParam คือตำแหน่งที่จะให้แทรกสตริง ส่วน lParam ไม่ใช้สำหรับเมสเสจนี้

สำหรับวิธีการส่งเมสเสจทำได้ 3 วิธีคือ

1. ใช้ SendMessage เป็นฟังก์ชันของวินโดวส์ใช้ได้กับหน้าต่างทุกชนิด โดยที่คอนโทรลก็เป็นหน้าต่างจึงนำมาใช้ได้ และใช้ได้ทั้งในหน้าต่างและในไดอะล็อกบ็อกซ์

2. ใช้ SendDlgItemMessage เป็นฟังก์ชันของวินโดวส์เช่นกัน เพื่อการส่งเมสเสจให้แก่คอนโทรล โดยเฉพาะใช้ได้ทั้งในหน้าต่างและในไดอะล็อกบ็อกซ์

3. ใช้ SendDlgItemMsg ซึ่งเป็นกิจกรรมของสกุล TDialog จึงใช้ได้แต่ในไดอะล็อกบ็อกซ์เท่านั้น

การสืบสกุลจาก TDialog

คงมีข้อสงสัยครับว่า เมื่อไรจะส่งเมสเสจจากหน้าต่าง และเมื่อไรจะส่งเมสเสจจากไดอะล็อกบ็อกซ์

สำหรับเทอร์โบปาสคาล จะส่งเมสเสจจากหน้าต่างเมื่อ ไม่มีสิ่งใดที่จะต้องดำเนินการในไดอะล็อกบ็อกซ์ คงใช้แต่บริการของวินโดวส์ ไม่ต้องสร้างผู้สืบสกุลจาก TDialog ใช้ TDialog โดยตรงในโปรแกรม

สื่อความที่ส่งให้จะเป็นสื่อความเมื่อเริ่มต้น เช่น การเซตปุ่มบางปุ่ม การให้ค่าแก่กรอบรายการ เป็นต้น แล้วนำค่าที่ ExecDialog ส่งกลับมาตรวจว่าผู้ใช้ยุติการทำงานกับไดอะล็อกบ็อกซ์ด้วยปุ่มใด เช่น OK หรือ Cancel นำไปดำเนินการตามที่ผู้ใช้ต้องการ ดูตัวอย่างการใช้ได้จากโปรแกรมในไฟล์ C:\TPW\OWLDEMOS\VDLGAPP.PAS

ทำคอนโทรลเป็นออปเจ็กต์

คอนโทรลในไดอะล็อกบ็อกซ์ไม่ใช่ออปเจ็กต์ แต่สามารถทำให้เป็นออปเจ็กต์ได้โดยใช้ InitResource อินิตคอนโทรล (คอนโทรลที่เป็นออปเจ็กต์แต่ต้นใช้ Init ในการอินิต) เพื่อการต่อเชื่อมคอนโทรลที่เป็นออปเจ็กต์ทางภาคของเทอร์โบปาสคาล เข้ากับคอนโทรลทางภาคของวินโดวส์

ปกติไม่จำเป็นต้องทำคอนโทรลให้เป็นออปเจ็กต์ จะทำให้คอนโทรลเป็นออปเจ็กต์เมื่อต้องการใช้กลไกในการถ่ายทอดข้อมูล (transfer mechanism) ที่เทอร์โบปาสคาลจัดไว้ ดูตัวอย่างได้จากโปรแกรมในไฟล์ C:\TPW\DOCDEMOS\TRANTEST.PAS

อีกเรื่องหนึ่งคือต้องการปรับปรุงคอนโทรลให้ทำหน้าที่มากกว่าที่กำหนดไว้เดิมเช่น เมื่อต้องการให้ผู้ใช้ป้อนข้อมูลเป็นเลขจำนวน และต้องการให้กรอบอีดิตเป็นผู้ตรวจ จึงต้องให้เป็นออปเจ็กต์และเป็นผู้สืบสกุลจาก TEdit ดูตัวอย่างได้จากโปรแกรมในไฟล์ VDLGAPP.PAS เช่นกัน

เพิ่มความเข้าใจต่อ

แต่ผมต้องการแสดงทางเดินของโปรแกรมอีกเช่นกันครับ ขอให้ลอกไฟล์ TSTDLG4.PAS เป็น TSTDLG5.PAS แก้โปรแกรมของกิจกรรม WndProC.cmDemo ให้เป็นดังรูปที่ 6 แล้วป้อนโปรแกรมของสกุล NumEditC.CanClose ตามรูปที่ 8

หลักการก็คือต้องการให้ผู้ใช้ป้อนข้อมูลที่กรอบ Edit เป็นตัวเลข เมื่อผู้ใช้เลือก OK หรือ Cancel จะทำงานที่ Canclose ของไดอะล็อกบ็อกซ์ทั้งคู่ เพราะเป็นโหมดเลสส์ (ที่ถูกต้องให้เป็นโมเดลไดอะล็อกบ็อกซ์)

CanClose ของไดอะล็อกบ็อกซ์ในฐานะหน้าต่างแม่จะเรียก CanClose ของหน้าต่างลูกได้แก่คอนโทรล (เฉพาะที่เป็นออปเจ็กต์) มาทำงานใน NumEditC.CanClose ซึ่งจะตรวจสอบว่าผู้ใช้ป้อนข้อมูลเป็นตัวเลขหรือไม่ ถ้าไม่ใช่จะแสดงข้อความเตือน

ถ้าผู้ใช้เลือกปุ่ม Yes แสดงว่าต้องการแก้ จึงให้เลือกข้อความเดิมทั้งหมดด้วย SetSelection ทำให้แสดงข้อความเป็นไฮไลต์ และ SetFocus ให้ตนเองเพื่อให้แอคตีฟ

สามารถใช้กับไดอะล็อกบ็อกซ์ที่สืบสกุลจาก TDialog ได้เช่นกัน เช่น TstDlgC ตามที่ผมใส่หมายเหตุไว้ในรูปที่ 6 แต่ต้องแก้ใน TstDlgC.Can-

```
  NumEditCP = ^NumEditC;
  NumEditC  = object(TEdit)
function  CanClose : Boolean;  virtual;
            end;


CONST  txtlen = 20;

FUNCTION  NumEditC.CanClose : Boolean;
VAR   dat : array[0..txtlen] of Char;
      num : Real;  res : Integer;
BEGIN
  MessageBox(HWindow,'CanClose','EDIT BOX',id_OK);
  GetText(dat,SizeOf(dat));  Val(dat,num,res);  CanClose := True;
  if  res <> 0  then
    if  MessageBox(HWindow,'Must be number want to correct',
                   'Error in Dialog Box',
                   mb_YesNo or mb_IconExclamation) = id_Yes  then
      begin  SetSelection(0, MaxInt);  SetFocus(HWindow);
             CanClose := False  end;
END;
```

รูปที่ 8 สกุล NumEditC

```
TYPE
  WndProCP = ^WndProC;
  WndProC  = OBJECT(TWindow)
  constructor  Init(apar : PWindowsObject; atitle : PChar);
  procedure  cmDemo(var msg : TMessage); virtual cm_first + cm_demo;
             END;


CONSTRUCTOR  WndProC.Init(apar : PWindowsObject; atitle : PChar);
BEGIN
  TWindow.Init(apar,atitle);
  RegisterType(RAddrCollect);  RegisterType(RAddr);
  attr.menu := LoadMenu(Hinstance, 'NMECRDMENU');
END;


PROCEDURE  WndProC.cmDemo(var msg : TMessage);
BEGIN
  Application^.ExecDialog(New(NameDlgCP, Init(@Self, 'NMECRDDLG')));
END;
```

รูปที่ 9 โปรแกรมของสกุล WndProC

Close ของเดิมเป็น CanClose:=TDialg.CanClose (และควรให้อยู่ที่บรรทัดสุดท้าย)

ที่ผมนำมาแสดงเพราะหาตัวอย่างดูได้ยาก (คือใช้ทั้งผู้สืบสกุลจาก TDialog และจากคอนโทรล เช่น TEdit) และผมงงอยู่นานเพราะในชั้นแรกผมเรียกใช้ TDialog.CanClose โดยตรง (ไม่ได้ให้ค่า) ปรากฏว่าผลการทำงานของ NumEditC.CanClose ไม่ส่งผลมาถึง ผู้ที่คุ้นเคยกับภาษาปาสคาลเมื่อมาใช้ฟังก์ชันอย่างภาษาซีก็อาจเผลอได้ครับ

เขียนโปรแกรมให้ไดอะล็อกบ็อกซ์

โปรแกรมตามตัวอย่างที่ผ่านมาผมยังไม่ถือว่าเป็นการเขียนโปรแกรมให้ไดอะล็อกบ็อกซ์ครับ เพราะเพียงเพื่อเสริมสร้างความเข้าใจเท่านั้น

เหตุที่เราต้องเขียนโปรแกรมให้ไดอะล็อกบ็อกซ์ เพราะคอนโทรลจะส่งเมสเสจให้แก่ไดอะล็อกบ็อกซ์เท่านั้น (ไม่ส่งให้หน้าต่าง) และเมื่อเราจำเป็นต้องดักจับเมสเสจจากคอนโทรลอย่างเช่นโปรแกรมนามบัตร ตามที่ได้กำหนดสกุล NameDlgC ในรูปที่ 12

ที่ผมยกเรื่องนี้มาเป็นตัวอย่างก็เพื่อให้เปรียบเทียบกับการเขียนโปรแกรมโดยใช้คอนโทรลเป็นออปเจ็กต์ในหน้าต่าง (ไมโครฯ 85) ข้อพิจารณาในการเขียนโปรแกรมจึงเหมือนกัน คือเพียงย้ายโปรแกรมที่อยู่ในหน้าต่างมาไว้ในไดอะล็อกบ็อกซ์ และผมได้ปรับปรุงโปรแกรมบ้างจึงขอกล่าวที่โปรแกรมโดยตรง

เนื้อหาของโปรแกรม

แต่เพื่อที่จะได้ไม่ต้องกลับไปอ่านตอนที่แล้วขณะนี้ ผมขอนำเนื้อหาของโปรแกรมมากล่าวโดยย่อ คือเมื่อเปิดใช้ไดอะล็อกบ็อกซ์ตามรูปที่ 11 จะเซตปุ่ม Enter ไว้เพื่อให้ผู้ใช้ป้อนข้อมูลในกรอบอีดิต เมื่อป้อนเสร็จให้กดปุ่ม Save เพื่อเก็บลงไฟล์ และนำชื่อไปแสดงที่กรอบคอมโบ

หากผู้ใช้เลือกปุ่ม Erata, Delete หรือ Search เพื่อการแก้ ยกเลิกหรือค้นหาข้อมูลตามลำดับ จะอ่านข้อมูลจากไฟล์ให้แก่กรอบคอมโบให้ผู้ใช้เลือก ซึ่งเมื่อเลือกแล้วหากต้องการให้นำออกแสดงให้เลือกปุ่ม Show เพื่อ

- แก้แล้วเลือกปุ่ม Save
- หรือถอนข้อมูล โดยเลือกปุ่ม Delete

สำหรับปุ่ม Clear เพื่อลบภาพในกรอบอีดิต และให้ยุติการทำงาน โดยเลือกปุ่ม OK

หมายเหตุ

เนื่องจากบทความในตอนนี้จำเป็นที่จะต้องอ้างถึงโปรแกรมเดิม คำว่าโปรแกรมเดิม ขอหมายถึงโปรแกรมในโปรแกรม NamCard3 ในไมโครฯ 85 โดยเฉพาะโปรแกรมของสกุล WndProC ครับ

Init และ Done ตามรูปที่ 13

- ใน Init เพียงแต่เรียกใช้ Init ของ TDialog เพราะต้องการ constructor เนื่องจากมีกิจกรรมที่เป็นเวอร์ชวล
- ใน Done ให้ยกเลิก fakefi ซึ่งเป็นไฟล์เทียมในหน่วยความจำ (การ

สร้างอยู่ใน SetUpWindow)

– การที่ให้มี Done ซึ่งความจริงไม่ต้องมี เพื่อให้ทดลองดูว่า ถ้าผู้ใช้ยุติการทำงานกับไดอะล็อกบ็อกซ์ด้วย Close ของคอนโทรลเมนูจะไม่ทำงานใน CanClose เช่นเมื่อป้อนข้อมูลแล้ว Close จะไม่เก็บลงไฟล์ ต้องยุติการทำงานโดยเลือกปุ่ม OK จึงจะมาทำงานที่ CanClose แตกต่างไปจากหน้าต่างธรรมดา

CanClose

อยู่ในรูปที่ 13 เช่น เป็นโปรแกรมเดิม เพื่อการเก็บข้อมูลไฟล์เทียมลงสตรีม (ไฟล์ในดิสก์)

SetUpWindow

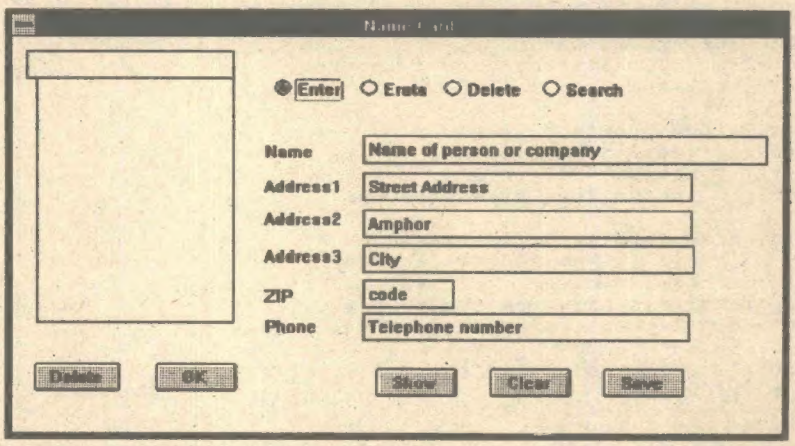
อยู่ในรูปที่ 14 ตอนต้นเป็นโปรแกรมเดิมเพื่อการโหลดข้อมูลจากสตรีมลงไฟล์เทียม ตอนท้ายเป็นโปรแกรมที่

– ส่งเมสเสจ bm_SetCheck ไปให้แก่ปุ่ม Enter ด้วยค่า 1 เพื่อเซตปุ่มนี้ทำให้สามารถแท็ปมาที่ปุ่มนี้ได้ ตามที่ตอนที่แล้วผมยังไม่ทราบและหากใช้คำสั่งนี้ในโปรแกรมเดิมจะแท็ปเข้ามาได้เช่นกัน

– ส่งเมสเสจ em_LimitText ไปให้แก่กรอบอีดิตทุกกรอบเพื่อจำกัดความยาวของสตริงในแต่ละกรอบ

```
TYPE
  FormFiCP = ^FormFiC;
  FormFiC  = object(TDialog)
    fakefi   : AddrCollectCP;
    select   : RadioT;
    loaded   : Boolean;
    tran     : PChar;
    finme    : FNameStr;
    fstid,lstid : Integer;
    fldsi    : AryIntP;

  constructor  Init(apar : PWindowsObject; atitle : PChar;
                    filenme : FNameStr;
                    firstid,lastid : Integer;
                    fieldsi : AryIntP):
  destructor  Done; virtual;
```

รูปที่ 10 ส่วนหนึ่งของสกุล FomFic ที่ให้ปรับปรุงจากสกุล NameDlgC



รูปที่ 11 ไดอะล็อกบ็อกซ์ที่ใช้เป็นแบบกรอกข้อมูล

รูปที่ 12

```
TYPE
  NameDlgCP = ^NameDlgC;
  NameDlgC  = object(TDialog)
    fakefi   : AddrCollectCP;
    select   : RadioT;
    tran     : AdrTrnT;
    loaded   : Boolean;

  constructor  Init(apar : PWindowsObject; atitle : PChar);
  destructor  Done; virtual;
  procedure  SetupWindow;  virtual;
  function   CanClose : Boolean; virtual;
  procedure  idEntr(var msg : TMessage); virtual id_first + id_entr;
  procedure  idErata(var msg : TMessage); virtual id_first + id_erata;
  procedure  idDel(var msg : TMessage); virtual id_first + id_del;
  procedure  idSrch(var msg : TMessage); virtual id_first + id_srch;
  procedure  idShwd(var msg : TMessage); virtual id_first + id_shwd;
  procedure  idClr(var msg : TMessage); virtual id_first + id_clr;
  procedure  idSave(var msg : TMessage); virtual id_first + id_save;
  procedure  idDelete(var msg : TMessage); virtual id_first + id_delete;
  procedure  DoLoadList;
  procedure  TranToEdit(var dat : AdrTrnT);
  procedure  TranFromEdit(var dat : AdrTrnT);
    end;
```

```
CONSTRUCTOR  NameDlgC.Init(apar : PWindowsObject; atitle : PChar);
BEGIN
  TDialog.Init(apar,atitle);
END;


DESTRUCTOR  NameDlgC.Done;
BEGIN
  Dispose(fakefi,Done);  TDialog.Done
END;


FUNCTION  NameDlgC.CanClose : Boolean;
VAR   adrstrm : TDosStream;
BEGIN
 CanClose := True;
 adrstrm.Init(adrFiName,stCreate);
 if  adrstrm.status <> 0  then ErrorInfo(adrstrm.status)
 else {if  fakefi^.count > 0  then } adrstrm.Put(fakefi);
 adrstrm.Done;
END;
```

รูปที่ 13

```
PROCEDURE NameDlgC.SetupWindow;
VAR   adrstrm : TDosStream;  idc : Integer;
      Hctrl : Hwnd;  sty : Word;
BEGIN
  TDialog.SetupWindow;
  adrstrm.Init(adrFiName,stOpen);
  if  adrstrm.status = 0  then
    fakefi := AddrCollectCP(adrstrm.Get)
  else
    begin  ErrorInfo(adrstrm.status);
           fakefi := New(AddrCollectCP,Init(100,10))  end;
  select := rentr;  loaded := False;  adrstrm.Done;
  SendDlgItemMsg(id_entr, bm_SetCheck,1,0);
  EnableWindow(GetDlgItem(HWindow,id_shwd),False);
  for  idc := id_nme to id_phne  do
    SendDlgItemMsg(idc,em_LimitText,fldlen[idc],0);
  TranToEdit(dumadr);
END;
```

รูปที่ 14

idEnter, idErata,

idDel และ idSrch

ทั้งสี่กิจกรรมนี้เป็นกิจกรรมสำหรับดักจับเมสเสจ เมื่อผู้ใช้เลือกปุ่ม Enter, Erata, Delete และ Search ตามลำดับ โดยที่ในโปรแกรมเดิมผมตรวจจับจากกรอบ Select แต่ในโปรแกรมใหม่นี้ต้องการแยกการทำงานดังโปรแกรมในรูปที่ 15 คือ

– DoLoadList เป็นกิจกรรมของสกุล NamDlgC เพื่อการโหลดข้อมูลจากไฟล์เทียมให้แก่กรอบคอมโบ

– สื่อความ cb_ResetContent ที่ส่งให้แก่กรอบคอมโบ เพื่อยกเลิกค่าในลิสต์และลบภาพในกรอบ

– EnableWindow เพื่อการเอนาเบิล คือให้ใช้เม้าส์และคีย์บอร์ดกับหน้าต่างได้กรณีนี้คือคอนโทรล เมื่อผ่านค่า True ไปให้ ถ้าผ่านค่า False ไปให้จะใช้ไม่ได้ และแสดงภาพคอนโทรลเป็นภาพจาง (gray) คือดิสเอเบิล

ผมนำมาใช้กับกรอบอีดิตที่ให้ป้อนชื่อโดยให้ดิสเอเบิลเมื่อผู้ใช้เลือกปุ่ม Erata เพื่อไม่ให้ผู้ใช้แก้ข้อมูลในกรอบนี้เนื่องจากใช้ฟิลด์นี้เป็นคีย์

และดิสเอเบิลปุ่ม Show เมื่อผู้ใช้เลือกปุ่ม Enter กับดิสเอเบิลปุ่ม Save เมื่อผู้ใช้เลือกปุ่ม Delete หรือ Search เพื่อไม่ให้เก็บลงไฟล์เป็นการซ้ำซ้อน โดยไม่ต้องสร้างเงื่อนไขในโปรแกรม ซึ่งผมเห็นว่าน่าจะเป็นวิธีเขียนโปรแกรมบนวินโดวส์ที่ถูกต้องกว่า

idShwd

ตอบสนองการเลือกปุ่ม Show ของผู้ใช้อยู่ในรูปที่ 16 โดยอ่านตำแหน่งของคีย์ คือชื่อที่ผู้ใช้เลือกจากกรอบคอมโบ โดยส่งเมสเสจ cb_GetCurSel ไปให้นำไปขอค่าจากไฟล์เทียม แล้วเลือกเฉพาะข้อมูลนำออกแสดงด้วย Tran-ToEdit

idClr

ลบข้อความในกรอบอีดิตทั้งหมดอยู่ในรูปที่ 16 โดยให้ข้อมูลเปล่าแก่ Tran-ToEdit

idSave

เก็บข้อมูลในกรอบอีดิตลงไฟล์ เมื่อผู้ใช้เลือกปุ่ม Save คงเป็นโปรแกรมเดิมที่เปลี่ยนมาใช้เมสเสจแทนตามรูป F มีเรื่องที่ยังไม่เคยกล่าวถึง คือ

– TranFromEdit อ่านข้อมูลจากกรอบอีดิตให้แก่ Tran

– ส่งเมสเสจ cb_AddString ไปให้กรอบคอมโบ โดยให้ค่าแอดเดรสของฟิลด์ name แก่ lParam ซึ่งเป็น Long-Int จึงต้องใช้วิธี typecast

idDelete

ถอนข้อมูลทั้งในกรอบคอมโบและในไฟล์เมื่อผู้ใช้เลือกปุ่ม Delete ใต้กรอบคอมโบ ตามโปรแกรมในรูปที่ 17 เขียนตามโปรแกรมเดิม มีเมสเสจใหม่คือ cb_DeleteString ที่ส่งไปให้กรอบคอมโบเพื่อถอนสตริง ณ ตำแหน่ง idx

DoLoadList

เป็นกิจกรรมสนับสนุนของสกุลนี้ อยู่ในรูปที่ 18 เพื่อการโหลดข้อมูลจากไฟล์เทียมมาให้แก่กรอบคอมโบ โดย

– ส่งสื่อความ cb_ResetContent ไปให้แก่กรอบคอมโบ เพื่อยกเลิกข้อมูลเดิมและลบภาพในกรอบนี้

– จากนั้นได้ให้อ่านข้อมูลจากไฟล์ครั้งละเรคอร์ด แยกฟิลด์ name ไปเก็บในกรอบคอมโบด้วยเมสเสจ cb_InsertString ณ ตำแหน่ง idx คือจะเป็นการเก็บต่อท้าย แต่จะจัดเรียงกันเพราะข้อมูลในไฟล์เก็บไว้แบบจัดเรียง

เรื่องนี้ความจริงแล้วผมต้องการเก็บลงกรอบคอมโบในแบบต่อท้าย โดยให้ค่า wParam เป็น –1 แทนที่จะเป็น idx ซึ่งทั้งในคู่มือของเทอร์โบปาสคาลและวินโดวส์ (Microsoft Windows Programmer's Reference) กล่าวไว้ตรงกันว่าจะเป็นการเก็บแบบต่อท้าย แต่ให้ค่า –1 ไม่ได้ เพราะในเทอร์โบปาสคาลเกิดกำหนดแบบข้อมูลของ wParam เป็น Word แทนที่จะเป็น Integer ทั้งใน SendDlgItemMsg และ SendDlgItemMessage

TranToEdit
TranFormEdit

ในโปรแกรมเดิมผมใช้ TransferBuffer ในการอ่านเขียนกับกรอบอีดิตเป็นกลุ่ม แต่ในโปรแกรมใหม่นี้ใช้ไม่ได้ เนื่องจากคอนโทรลไม่ใช่ออปเจ็กต์ ใช้สื่อความ EnableTransfer ไม่ได้ จึงต้องเขียนโปรแกรมดำเนินการเองตามโปรแกรมของสองกิจกรรมนี้ในรูปที่ 18 กล่าวคือ

– เมสเสจ wm_SetText เป็นเมสเสจให้ค่าเท็กซ์แก่หน้าต่างทุกชนิด เช่นใช้เป็นชื่อหน้าต่าง สำหรับคอนโทรลขึ้นอยู่กับชนิดของคอนโทรล กรอบอีดิตคือข้อความที่แสดงให้ผู้ใช้แก้หรือป้อนใหม่ โดยให้แอดเดรสของสตริงแก่ lParam

– เมสเสจ wm_GetText ดำเนินการตรงข้ามคือ อ่านเท็กซ์จากหน้าต่างให้แก่สตริง โดยต้องส่งความยาวของสตริงไปให้ WParam ด้วย

```
PROCEDURE  NameDlgC.idEntr(var msg : TMessage);
BEGIN
  SendDlgItemMsg(id_combo, cb_ResetContent, 0, 0);
  select := rentr;  loaded := False;
  EnableWindow(GetDlgItem(HWindow,id_nme),True);
  EnableWindow(GetDlgItem(HWindow,id_save),True);
  EnableWindow(GetDlgItem(HWindow,id_shwd),False);
END;

PROCEDURE  NameDlgC.idErata(var msg : TMessage);
BEGIN
  if  not loaded  then  DoLoadList;
  select := rerata;
  EnableWindow(GetDlgItem(HWindow,id_shwd),True);
  EnableWindow(GetDlgItem(HWindow,id_save),True);
  EnableWindow(GetDlgItem(HWindow,id_nme),False);
END;

PROCEDURE  NameDlgC.idDel(var msg : TMessage);
BEGIN
  if  not loaded  then  DoLoadList;
  select := rdel;
  EnableWindow(GetDlgItem(HWindow,id_shwd),True);
  EnableWindow(GetDlgItem(HWindow,id_nme),True);
  EnableWindow(GetDlgItem(HWindow,id_save),False);
END;

PROCEDURE  NameDlgC.idSrch(var msg : TMessage);
BEGIN
  if  not loaded  then  DoLoadList;
  select := rsrch;
  EnableWindow(GetDlgItem(HWindow,id_shwd),True);
  EnableWindow(GetDlgItem(HWindow,id_nme),True);
  EnableWindow(GetDlgItem(HWindow,id_save),False);
END;
```

รูปที่ 15

```
PROCEDURE  NameDlgC.idShwd(var msg : TMessage);
VAR  tem : AddrCP;  idx : Integer;
BEGIN
  idx := SendDlgItemMsg(id_combo, cb_GetCurSel, 0, 0);
  if  idx < 0  then  Exit;
  tem := fakefi^.At(idx);  tem^.GetData(tran);
  TranToEdit(tran);
END;

PROCEDURE  NameDlgC.idClr(var msg : TMessage);
BEGIN
  TranToEdit(nulladr);
END;
```

รูปที่ 16

– สำหรับการให้ค่าแอดเดรสของสตริง ผมเริ่มโดยให้ aptr ชี้ที่ต้นเรคอร์ด แล้วบวกด้วยขนาดของฟิลด์ในการอ่านฟิลด์ต่อ ๆ ไป ที่ต้องบวกเพิ่มด้วย 1 เพราะได้กำหนดสตริงไว้เป็น [0..n] โดยที่ n คือความยาวของสตริง ที่เก็บไว้ในอะเรย์ fldlen

โปรแกรมทั้งหมด

ในการป้อนโปรแกรมขอให้ถือว่าเป็นการปรับปรุงโปรแกรมเดิมจะดำเนินการได้ดังนี้ครับ

1. ก๊อบปี้ไฟล์ NMECRD3.PAS เป็น NMECRD4.PAS แล้วแก้ชื่อโปรแกรมเป็น NameCard4

2. ลบการกำหนดสกุล WndProC และโปรแกรมของสกุลนี้ทั้งหมด แล้วป้อนโปรแกรมตามรูปที่ 12 ถึงรูปที่ 19

หรือเปลี่ยนชื่อสกุลจาก WndProC เป็น NameDlgC ทุกแห่ง แล้วแก้โปรแกรมให้เป็นตามรูปที่ 12 ถึงรูปที่ 19

3. ลบการกำหนดค่าคงตัวที่ขึ้นต้นด้วย id ทั้งหมด

4. ป้อนโปรแกรมตามรูปที่ 19 ไว้ตอนต้นของโปรแกรม

5. แก้คำว่า TCollection เป็น AddrCollectC และ PCollection เป็น AddrCollectCP ในที่ทุกแห่ง แล้วป้อนโปรแกรมตามรูปที่ 20 คือเปลี่ยนจากสกุล TCollection เป็น AddrCollectC (ซึ่งเรื่องนี้ควรจะเป็นเช่นนี้ในตอนที่แล้ว ไม่ทราบว่าผมทำตกหล่นไปได้อย่างไรครับ)

6. ป้อนโปรแกรมของสกุล WndProC ใหม่ตามรูปที่ 9

จะได้โปรแกรมที่พร้อมสำหรับทดลองรันครับ (ผมได้ฝากโปรแกรมนี้มาทางซอฟต์เมล์ด้วย)

```
PROCEDURE  NameDlgC.idSave(var msg : TMessage);
VAR   idx : Integer;
BEGIN
  TranFromEdit(tran);
  if  tran.name = ''  then  Exit;
  if  select = rentr  then
    begin  idx := SendDlgItemMsg(id_combo, cb_AddString, 0,
                                 LongInt(@tran.name));
           fakefi^.Insert(New(AddrCP, Init(tran)))  end
  else
    begin  idx := SendDlgItemMsg(id_combo, cb_GetCurSel, 0, 0);
           fakefi^.AtPut(idx, New(AddrCP, Init(tran)))  end;
  TranToEdit(nulladr);
END;


PROCEDURE  NameDlgC.idDelete(var msg : TMessage);
VAR   idx : Integer;
BEGIN
  if  select <> rdel  then  Exit;
  idx := SendDlgItemMsg(id_combo, cb_GetCurSel, 0, 0);
  if  idx >= 0  then
    begin  SendDlgItemMsg(id_combo, cb_DeleteString, idx, 0);
           fakefi^.AtDelete(idx)  end;
END;
```

รูปที่ 17

เพิ่มเติม

ตามที่กล่าวกันว่าประโยชน์ของรีซอร์สคือ เราสามารถแก้รูปแบบได้โดยไม่กระทบกระเทือนถึงโปรแกรมในไฟล์ .EXE ทดลองดูได้ครับโดย

– เปิดใช้ WRT

– เลือกปุ่ม Open

– เลือกไฟล์เป็น NMECRD4.EXE

– เลือก NMECRDDLG

– เลือกปุ่ม Edit

– สร้างปุ่ม Cancel ให้เป็นปุ่มควบคุมมีค่าไอดีเป็น 2

– เลือกรายการ File/Save

แล้วใช้ File Manager รันโปรแกรม NMECRD4.EXE ใหม่ จะได้ปุ่ม Cancel เพิ่มขึ้น ทำหน้าที่ได้สมบูรณ์ทุกประการ

ที่เราทำเช่นนี้ได้ก็เพราะเรารู้จักโปรแกรมนี้ครับ และรู้ด้วยว่าการทำงานของปุ่ม Cancel มีค่าไอดีเป็น 2 มีโปรแกรมสำหรับดำเนินการอยู่แล้ว

ถ้าไม่รู้จักโปรแกรม สิ่งที่ทำได้แน่คือ เปลี่ยนชื่อต่าง ๆ ย้ายตำแหน่งของกรอบและปุ่มต่าง ๆ และอย่างเช่นของโปรแกรมนี้ (และโปรแกรมอื่น ๆ ด้วย) เปลี่ยนชนิดของกรอบคอมโบจากชนิดธรรมดาเป็นชนิดที่โดยปกติไม่แสดงกรอบคอมโบจากชนิดธรรมดา เป็นชนิดที่โดยปกติไม่แสดงกรอบรายการ ส่วนการที่จะไปยกเลิกกรอบและปุ่มของเดิมหรือกำหนดขึ้นใหม่เป็นเรื่องที่อาจทำไม่ได้ครับ

เปรียบเทียบ

จากไดอะล็อกบ็อกซ์ทั้งสามชนิดและหน้าต่างอย่างไดอะล็อกบ็อกซ์ (คือหน้าต่างที่มีคอนโทรลเป็นออปเจ็กต์) ผมคิดว่างานที่เหมาะสมของแต่ละชนิดคือ

โมเดลไดอะล็อกบ็อกซ์ เหมาะกับ

งานที่รอคำตอบจากผู้ใช้ ถ้าไม่ได้คำตอบโปรแกรมจะทำงานต่อไปไม่ได้ เช่น เมื่อผู้ใช้ต้องการเปิดไฟล์ เราจะต้องให้ผู้ใช้ป้อนชื่อไฟล์ หรือไม่เช่นนั้นก็ให้ยกเลิกการเปิดไฟล์ จึงจะไปทำงานอื่นได้

โหมดเลสส์ไดอะล็อกบ็อกซ์ เมื่อต้องการให้ผู้ใช้ผละไปทำงานอื่นได้ด้วย เช่นค้นหาคำหรือข้อความมาแสดงให้ผู้ใช้ลอกไปใช้ อย่างเช่น เมื่อเลือกรายการ File/Search ของ File Manager หรือเมื่อสั่งงานพิมพ์เช่นใน Paintbrush

ไดอะล็อกบ็อกซ์อย่างหน้าต่าง ในคู่มือของเทอร์โบปาสคาลให้ใช้เป็นหน้าต่างหลักของแอพพลิเคชันที่ไม่มีหน้าต่างอื่น เช่นเครื่องคิดเลข (ไฟล์ C:\TPW\OWLDEMOS\CALC.PAS) แต่ผมคิดว่านำไปใช้งานใดก็ได้ ที่ต้องการโหมดเลสส์ไดอะล็อกบ็อกซ์ ที่เราต้องการจัดรูปแบบของเคอร์เซอร์ให้ใหม่ หรือต้องการให้มีไอคอนเอง (ซึ่งก็ควรจะมีไดอะล็อกบ็อกซ์แบบนี้หลายบาน)

หน้าต่างอย่างไดอะล็อกบ็อกซ์ ตามคำแนะนำของเทอร์โบปาสคาล ให้ใช้คอนโทรลที่เป็นออปเจ็กต์ในหน้าต่างเมื่อมีคอนโทรลไม่มากนัก แต่เมื่อนำมาใช้ในโปรแกรมนามบัตรก็ไม่ได้เสียเวลาในการจัดรูปแบบมากนัก ที่สำคัญคือเมื่อไม่มีรีซอร์สทำให้ไม่สามารถจัดรูปแบบไดอะล็อกบ็อกซ์ในไฟล์ .EXE ใหม่ได้

```
PROCEDURE  NameDlgC.DoLoadList;
VAR   adr : AddrCP;  idx : Integer;
BEGIN
  SendDlgItemMsg(id_combo, cb_ResetContent, 0, 0);
  for  idx := 0 to fakefi^.Count-1  do
    begin  adr := fakefi^.At(idx);
           SendDlgItemMsg(id_combo, cb_InsertString, idx,
                                    LongInt(adr^.GetName))  end;
  loaded := True;
END;


PROCEDURE  NameDlgC.TranToEdit(var dat : AdrTrnT);
VAR   idc : Integer;  aptr : PChar;
BEGIN
  aptr := PChar(@dat);
  for  idc := id_nme to id_phne  do
    begin  SendDlgItemMsg(idc, wm_SetText, 0,
                                    LongInt(aptr));
           aptr := aptr + fldlen[idc] + 1  end;
END;


PROCEDURE  NameDlgC.TranFromEdit(var dat : AdrTrnT);
VAR   idc : Integer;  aptr : PChar;
BEGIN
  aptr := PChar(@dat);
  for  idc := id_nme to id_phne  do
    begin  SendDlgItemMsg(idc, wm_GetText, fldlen[idc],
                                    LongInt(aptr));
           aptr := aptr + fldlen[idc] + 1  end;
END;
```

รูปที่ 18

```
PROGRAM  NameCard4;   { NMECRD4.PAS }

USES  Strings,WinTypes,WinProcs,WObjects,StdDlgs;


{$R NMECRD.RES}

CONST
  cm_demo = 201;
  id_combo = 101;     id_entr  = 102;     id_erata = 103;
  id_del   = 104;     id_srch  = 105;     id_sel   = 106;
  id_nme   = 113;     id_adr1  = 114;     id_adr2  = 115;
  id_adr3  = 116;     id_zip   = 117;     id_phne  = 118;
  id_shwd  = 119;     id_clr   = 120;     id_save  = 121;
  id_delete = 122;


TYPE
  RadioT = (rentr,rerata,rdel,rsrch);


CONST  nmelen = 39;  adr1len = 29;  adr2len = 19;
       ziplen = 7;   phnelen  = 29;
       fldlen : array[id_nme..id_phne] of Integer =
         (nmelen,adr1len,adr1len,adr2len,ziplen,phnelen);
```

รูปที่ 19

แบบกรอกข้อมูล

สกุล NameDlgC อาจสร้างให้เป็นสกุลกลางคือ ใช้เป็นแบบกรอกข้อมูลได้ทั่วไปโดยไม่ต้องเขียนโปรแกรมใหม่อีก คงเปลี่ยนแค่รีซอร์สของไดอะล็อกบ็อกซ์ เช่น ใช้ในการเก็บข้อมูลของหนังสือซึ่งจะมีฟิลด์ ชื่อหนังสือ ชื่อผู้แต่ง ชื่อวิชา สำนักพิมพ์ ฯลฯ คือจะมีจำนวนฟิลด์ และกรอบอีดิตไม่เท่ากัน

ในการนี้จำเป็นต้องปรับปรุงโปรแกรมหลายแห่งให้ใช้งานได้ทั่วไป (ไม่

เจาะจง) แต่ผมจะขอกล่าวเฉพาะจุดที่สำคัญเพื่อให้เห็นแนวทางคือ

1. จะต้องสร้างสกุลนี้เป็นยูนิต

2. ควรเปลี่ยนชื่อสกุลจาก NameDlgC ให้เป็นชื่อกลาง ๆ เช่น FormFiC

3. เพิ่มฟิลด์ข้อมูลและพารามิเตอร์ดังแสดงในรูปที่ 10 โดยที่

– finme และ filename คือชื่อไฟล์ที่จะใช้เก็บข้อมูล

- fstid และ firstid คือค่าไอดีของกรอบอีดิตที่จะเป็นข้อมูลของฟิลด์แรก

- lstid และ lastid คือค่าไอดีของกรอบอีดิตที่จะเป็นข้อมูลฟิลด์สุดท้าย โดยที่ค่าไอดีของกรอบอีดิตตั้งแต่ฟิลด์แรกถึงฟิลด์สุดท้ายจะต้องเรียงกัน และเรียงลำดับตามฟิลด์ของเรคอร์ด

– fldsi และ fieldsi คือพอยน์เตอร์ไปยังอะเรย์ของ Integer ที่เก็บค่าขนาดของฟิลด์แต่ละฟิลด์ ดังเช่น fldlem ในรูป X1 แต่กรณีนี้จะต้องเป็นอะเรย์ [0..n] เมื่อ n คือจำนวนฟิลด์ลบ 1

4. ใน Init ให้

– ให้ค่าตัวแปรที่คู่กัน

– คำนวณขนาดของเรคอร์ดจากขนาดของฟิลด์ใน fldsi นำไปจองหน่วยความจำให้แก่ tran

5. ในโปรแกรมทุกแห่ง ให้

– เปลี่ยนชื่อตัวแปรจาก adrFiName เป็น finme

– ยกเลิกการใช้ dumadr และ nulladr ให้เคลียร์ค่าใน tran ด้วย #0 แล้วนำมาใช้แทน

– ในวงรอบ for ให้เปลี่ยนจากของเดิม id_nme to id_phme เป็น fstid to lstid และปรับค่าอื่นตาม

6. ในการใช้งาน อาจใช้อินสแตนซ์จากสกุลนี้มากกว่า 1 ตัว เช่น ร้านวิดีโออาจใช้เป็นบัญชีสมาชิก 1 ตัว และบัญชีหนังอีก 1 ตัว

```
  AddrCollectCP = ^AddrCollectC;
  AddrCollectC = OBJECT(TSortedCollection)
    constructor  Init(alimit,adelta : Integer);
    constructor  Load(var S : TStream);
    procedure    Store(var S : TStream);
    function  KeyOf(item : Pointer) : Pointer; virtual;
    function  Compare(key1,key2 : Pointer) : Integer; virtual;
                END;

CONSTRUCTOR  AddrCollectC.Init(alimit,adelta : Integer);
BEGIN   TsortedCollection.Init(alimit,adelta)  END;

CONSTRUCTOR  AddrCollectC.Load(var S : TStream);
BEGIN
  TSortedCollection.Load(S)
END;

PROCEDURE  AddrCollectC.Store(var S : TStream);
BEGIN
  TSortedCollection.Store(S)
END;

FUNCTION  AddrCollectC.KeyOf(item : Pointer) : Pointer;
BEGIN
  KeyOf := @AddrCP(item)^.adrrcrd.name
END;

FUNCTION  AddrCollectC.Compare(key1,key2 : Pointer) : Integer;
VAR    keya,keyb : PChar;  res : Integer;
BEGIN
  Compare := StrIComp(PChar(key1), PChar(key2))
END;

CONST
  RAddrCollect : TStreamRec =
    (ObjType : 202;
     VMTlink : Ofs(TypeOf(AddrCollectC)^);
     Load    : @AddrCollectC.Load;
     Store   : @AddrCollectC.Store);
```

รูปที่ 20

ผมคิดว่าจุดใหญ่ ๆ คงมีเท่านี้ครับ และการเขียนโปรแกรมแบบนี้ ผมถือว่าเป็นการเขียนโปรแกรมในแบบโอโอพีในแง่ที่ทำให้สามารถใช้งานได้ทั่วไป ขอให้ทดลองดูครับ

ยังมีต่อ

เรื่องของไดอะล็อกบ็อกซ์มีเพียงเท่านี้ ที่ผมกล่าวว่ายังมีต่อเพราะยังไม่ได้นำไดอะล็อกบ็อกซ์สำเร็จรูปที่เตรียมไว้ในเทอร์โบปาสคาลมาใช้ แต่ตอนต่อไปต้องเปลี่ยนเป็นเรื่องอื่นก่อนครับ (ได้ฝากโปรแกรมที่กล่าวถึงในตอนนี้มาทางซอฟต์เมล์ด้วย)